GEO SENSITIVE WORD DISCOVERY

A Thesis

by

HAIPING XUE

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	James Caverlee
Committee Members,	Xia Hu
	Jim Ji
Head of Department,	Dilma Da Silva

May 2019

Major Subject: Computer Science

ABSTRACT

Among geolocation related information, in particular, the geo-sensitive word is one of the most critical components. A geo-sensitive word can be a word or phrase for a landmark in the city or county name, abbreviation sports team names in the city, common words or phrases with special meanings in local regions. In this thesis, we propose and evaluate an effective and efficient framework for discovering geo-sensitive words hidden in tweets. This framework overcomes the lack of dataset and embedding alignment problem. There are three key contributions in the proposed framework: (i) a publicly-available dataset containing geo-tagged English tweets from 27 cities in the United States; (ii) a concrete approach to align separately trained word embeddings with Orthogonal Procrustes; (iii) and a well-rounded evaluation framework for geo-sensitive words. The system discovers over 3000 geo-sensitive words in three cities and successfully classified these words into corresponding cities with a 95.32% high accuracy. We also find two key factors that post an impact on the classification performance: (i) feature vector dimension; and (ii) proper learning algorithm.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor James Caverlee and Professor Xia Hu of the Department of Computer Computer Science & Engineering and Professor Jim Ji of the Department of Electrical & Computer Engineering.

Funding Sources

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Geolocation is the identification or estimation of the real-world geographic location of an object. Software applications usually collect it from a mobile phone or internet-connect computer. Geolocation is essential across localized services and location-based social networks. A few successful examples of leveraging geolocation information include Facebook's regional advertisement and local events discovery, Google Maps' navigation and Explore Nearby, Uber and Lyft's core business: ridesharing. The success of leveraging geolocation creates new opportunities for new exciting purposes, such as the local marketplace, finding local friends, restaurants and entertainment facilities, geolocation powered augmented reality games and so on.

Among geolocation related information, in particular, the *geo-sensitive word* is one of the most critical components. A geo-sensitive word is a word or a phrase that contains local information in it, such a word or phrase for a landmark in the city or county name, abbreviation sports team names in the city, common words or phrases with special meanings in local regions. With these derived words, we can find and categorize points of interest, predict the location of user-generated text data, thus improving the accuracy of local search and personalized recommendation results.

However, while there have been some previous efforts to identify geo-sensitive words [1, 2, 3, 4], there remain a number of critical challenges:

- First, there are no existing datasets for the discovery of geo-sensitive words, meaning that we have to build a dataset that contains abundant sentences and locations for our training algorithm and analysis.
- Second, even if we could identify appropriate representations of different words in different places, it is unclear how to *align* these representations for direct comparisons. For example, it is unable to perform direct cosine similarity on feature vectors from different cities since they are not in the same coordinate system.
- Finally, there are challenges in evaluating the quality of methods for finding geo-sensitive

words. How can we be sure that our discovered words are high quality and can be applied to find geolocation related information in real world applications.

Towards overcoming these challenges, this thesis proposes an effective and efficient framework for discovering geo-sensitive information hidden in tweets. Our approach builds on the well-known word2vec approach, an efficient and effective neural network model for learning vector representations of words and phrases from very large data sets [7, 5]. Specifically, this thesis makes the following contributions:

- We create a publicly-available dataset containing geo-tagged English tweets from 27 cities in the United States. This dataset contains 4.7 millions tweets. We also detect phrases in these tweets.
- We show how to learn geo-specific word representations using word2vec, and how to align these different representations so that word feature vectors from separately trained Word embeddings can perform direct comparisons and classification tasks.
- We propose an evaluation framework for geo-sensitive words, wherein we build 4 different classifiers to perform a simple location prediction for discovered geo-sensitive words. By inspecting the performance of these classifiers, we confirm that our framework is able to identify geo-sensitive words and capture local information in their corresponding feature vectors.

The rest of this thesis is organized as follows. First, we introduce the related work in Chapter 2. Then we describe how to create our dataset based on a billion tweets in Chapter 3. In Chapter 4, we walk through our geo-sensitive word discovery approach. After identifying these words, we exhibit the proposed evaluation framework in Chapter 5. Finally, conclusion and future work are discussed in Chapter 6.

2. RELATED WORK

Study on the geographical estimation of web users has attracted many scholars in the last few decades. These researches cover a wide range of topics including finding local experts on Twitter [5], geo-locating a Facebook user based on located users [6], geo-locating Twitter users with their tweets [1], and so on. The recent success in Word2Vec related researches has inspired us to adopt this approach into geographical estimation studies. In this chapter, we highlight related work along three dimensions: wrong embedding, topically constrained word embedding, and content-based approach for geo-locating Twitter users.

2.1 word2vec

Word embedding, a useful and versatile tool for NLP (nature language processing) research, is a learned representation for text corpus where words with close meaning have similar representations [7]. Compared to NLP system, one of the biggest advantages of word embedding is to find relationships that may exist between the individual symbols instead of treating them as discrete atomic symbols. This enables word embedding to capture the context of a word in a document, the semantic and syntactic similarity, the relationships with other words. A few novel word embeddings include admirable vector space model [8], GloVe (Global Vectors for Word Representation) [9], Word2Vec [10] and etc. A handful of tremendous applications are built on top of the emergence and perfection of these innovative word embeddings, for instance, long short-term memory (LSTM) networks for language modeling [11], machine translation [12], text summarization [13] and named entity recognition [14]. Word2Vec is an efficient neural network model for learning state-of-the-art vector representation of words and phrases from very large data sets [15, 10]. It has gained huge attention in recent years and has been applied to a variety of natural language processing and machine learning related tasks such as sentiment analysis in Twitter [16] and image classification [17]. Word2vec is a three-layer neural network which contains a input layer, a hidden layer and an output layer. Its input is usually all words' one-hot vectors in a large text corpus.

Its output is the probability distribution of target words. It trains words against other words that neighbor them in the input corpus with CBOW (continuous bag of words) or skip-gram.

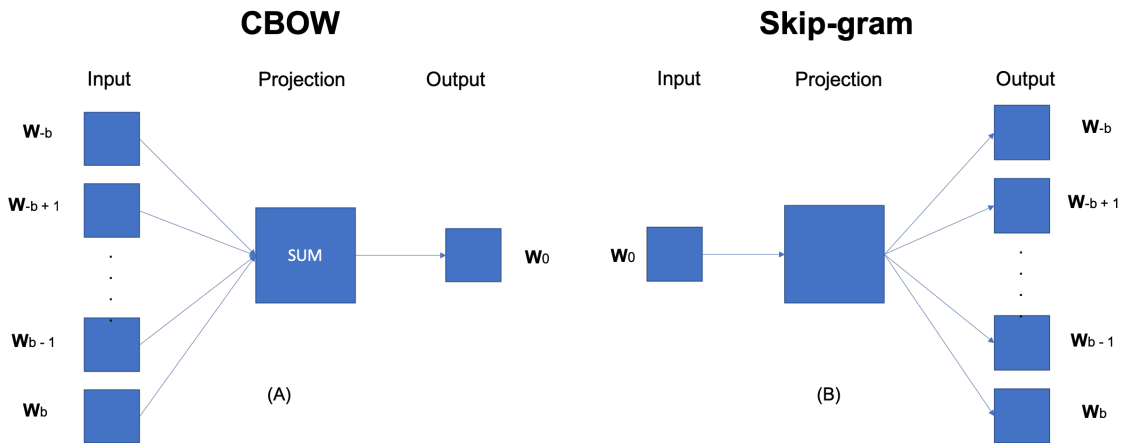

Figure 2.1: Illustration of the Skip-gram and CBOW model

The skip-gram model in Figure 2.1 (A) is for prediction of contextual words based on the center word. For a document of M words, given a word w_i we want to maximize the following objective function:

$$\frac{1}{M}\sum_{d=1}^{D}\sum_{-b\leq k\leq b, k\neq 0}\log p(w_{d+k}|w_d) \tag{2.1}$$

where b is the hyperparameter for window size, d is the position of the word, and $p(w_{d+k}|w_d)$ is a probability derived by softmax function. However, it is quite inefficient to compute the softmax function since calculation of the gradient of softmax is expensive. Hierarchical softmax objective function or negative sampling is used to deal with this issue. To maximize the conditional log-likelihood of this model, the Hierarchical softmax adopts a Huffman tree to minimize calculation needed. Negative sampling, on the other hand, minimizes computations during the training process through sampling a fix number of negative instances for the target word rather than sampling the entire vocabulary in the model.

In contrast to skip-gram model, continuous Bag of Words model in Figure 2.1 (B) does the

opposite job: predicting the target word given its nearby words $w_{-b}, w_{-b+1}, ..., w_{b-1}, w_b$. The prediction result is not affected by the order of context words. Therefore, Word2Vec model is more meaningful for discovering words with similar semantic meaning than words' syntactic properties. In this work, we choose to use CBOW model.

A well-trained word embedding model tends to embed words with similar meanings closer in the vector space. For example, "cat" and "dog" will be close in the vector space of trained model since they are all pets [10]. With the advanced word embeddings obtained from Word2Vec, we can not only find relationships that may exist between the individual symbols but also perform basic vector algebra between two word vectors. For example, vector("King") - vector("Man") + vector("Woman") resulting in vector("Queen").

2.2 Topically Constrained Word Embedding

Recent researches illustrate that word embeddings derived from Word2Vec [12] can effectively perform natural language processing tasks such as word similarity comparison. The general approach for these tasks is to train a global word embedding where each word has their own fixed representation in a word feature vector. However, the usage of words and phrases can change rapidly based on different topics. For example, the word "pointer" means a breed of hunting dog in animal-related topics while it means a variable to an address in a memory location where a value related to it is stored. Even with the local windows in Word2Vec, we still have a high possibility to snap only representations of the dominant topics in the text corpora.

In Diaz's recent work related to query expansion with Word2Vec [18], he proposed that many tasks can benefit from learning a topically constrained word embedding. He started by illustrating the word-context pair distribution in topic-constrained text corpora. In his text corpus, he found a substantial change for distributions of words in different subtopics. Then in his experiment, the locally trained word embedding outperforms the globally trained embedding in his query expansion tasks by a large margin. This proves that topic-constrained word embedding can provide a better similarity result in many tasks. Based on Diaz's recent work we believe separately trained Word2Vec models with different cities' text corpus will be able to learn a better representation

that captures geo-sensitive informations in words and phrases. In addition to this, we propose an approach to enable direct comparisons on feature vectors from Word2Vec models trained from different cities.

2.3 Content-Based Approach for Geo-locating Twitter Users

A handful of studies related to geo-location prediction problems on social networks have been done by researchers in the last few years. Various applications have been proposed to study them, such as finding local experts for the recommendation on Twitter [5], geo-locating users based on their tweets [1], and prediction of individuals locations based on geo-tagged users [6]. A few common approaches are adopted for these problems. One traditional approach is to use external resources from databases or a user's IP address or a gazetteer as ground truth to coordinate the target [19]. With the growing interests in social networks, a novel approach, content-based approach, is proposed by Cheng [1] and has gained huge attention. Compared to traditional approaches, Cheng's method exhibits an essential advantage: no requirements for any extra geolocation cues from another resource. The geo-location information is solely derived from the users' profiles. In the following studies, scholars enhanced his approach by adding more elements to it. Chandra's work improved Cheng's work by leveraging interactions between related tweets [20]. Kinsella's work increased prediction accuracy for the origin of a tweet with geolocation information from tweets rather than from the users' profiles [21]. In these works, each user is assigned to one city based on their tweets. All these content-based methods share two essential features: probability distribution models for terms in different cities and an estimator for locating the user with their words used in their tweets.

In our work, we focus on the improvement of the probability distribution models in these content-based approaches. In Cheng's approach [1], the probability distribution of terms is calculated based purely on word frequency. It ignores each word's association with other words in the probability distribution model. By adding words' association into the probability distribution model, we are able to use the co-occurrence of associated terms as a strong signal to indicate where the target tweet is from. For example, there is a high chance that the target tweet is from Houston if

the term "Rockets" and the term "NBA" occur in the tweets concurrently. To capture such kind of connections in our probability model, we use Word2Vec [10] to achieve it. By training our model using Word2Vec, we can observe words that tends to co-occur and spot associations that can be served as a strong signal to indicate the location of a target tweet.

3. CREATING A DATASET OF GEO-SENSITIVE WORDS

In this chapter, we construct our dataset from a billion tweets provided by Cheng [22]. In Cheng's dataset, there are tweets from places all over the world in different languages. For our work, we would like to scale down our research to English tweets in major cities in the United States. We collect, separate, and preprocess English tweets into 27 city text corpora which contain only clean text data.

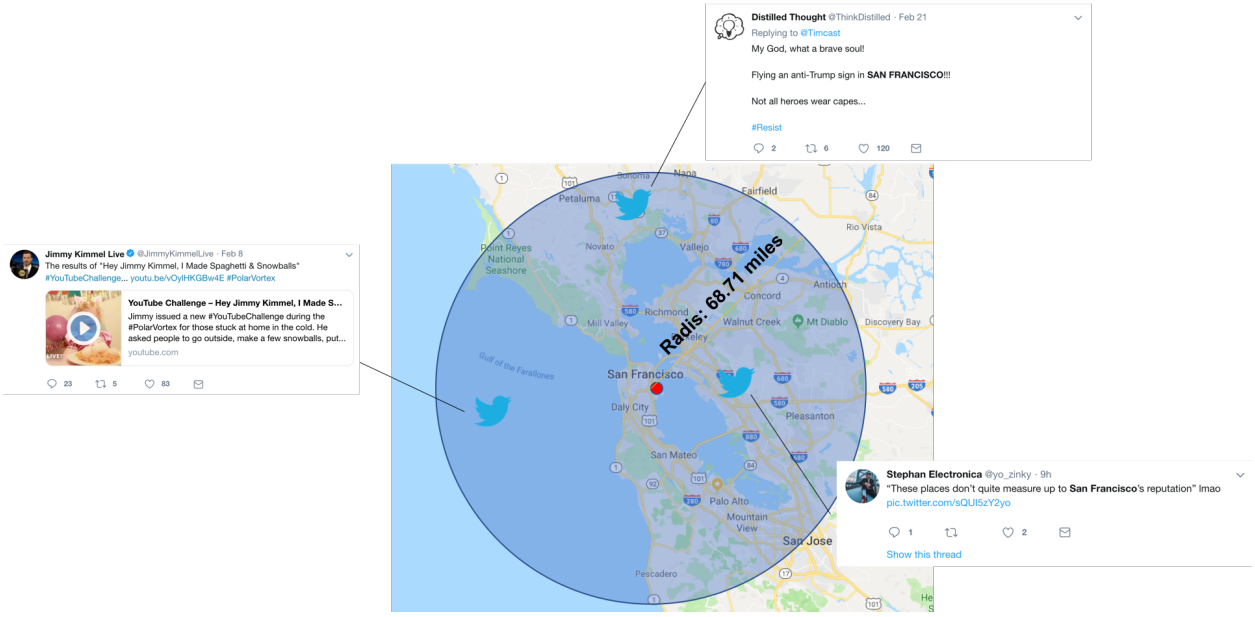

Figure 3.1: The circle used to collect tweets in San Francisco

There are about a billion geotagged tweets provided Cheng [22] from Twitter using the Twitter Streaming API from January 2014 to April 2014, and September 2015 to November 2015. Each tweet in this collection contains a coordinate with latitude and longitude indicating the geolocation where the tweet is created. Based on the coordinate, we extract tweets belong to 27 different cities in the United States and separated them into different files. To gather more tweets in a city, we do not use the city name in the tweet to verify if a tweet is in a city, instead, we collect tweets in a

circle defined by a center coordinate of each city and a radius (0.87544195). The radius we used is based on latitude and longitude. The equivalent length of it is approximately 60 miles. Figure 3.1 illustrates how the circle is defined in San Francisco. We obtain 4.7 millions of tweets in total. The number of tweets in each city are shown in Figure 3.2.

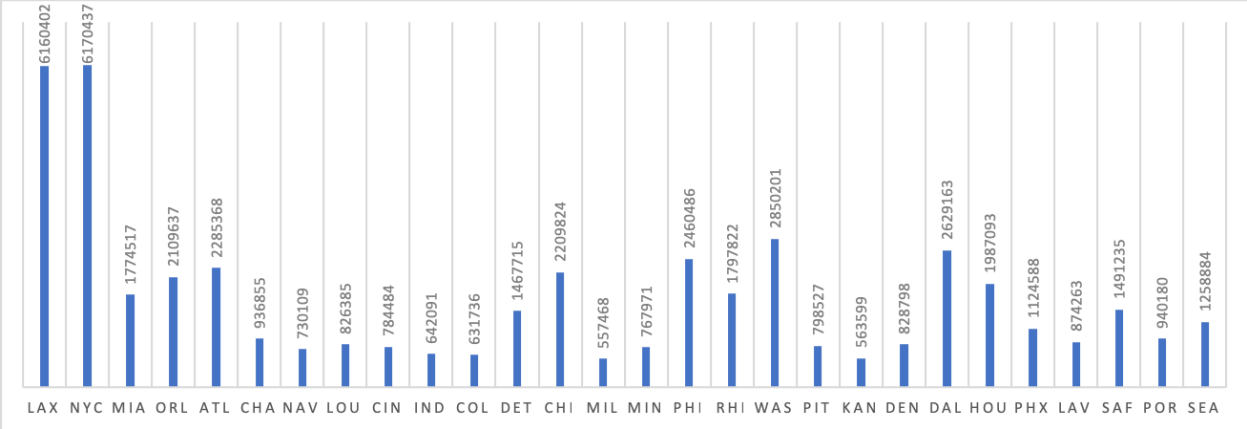

Figure 3.2: Tweets count for each city

In the next step, we separate all tweets into different text corpora based on our city labels and we only keep text data in the tweets. This helps us shrink the size of the file from 200GB to 2GB. With the much smaller text corpus, we can preprocess our text corpus and train our model faster.

By looking at the tweets we collected at the point, we spot many non-English words, same words with different cases and website links in tweets. This introduces a few problems to Word2Vec training. Since Word2Vec uses words' past appearance and words' associations with other words, more occurrences of the same word will help Word2Vec learn and predict this word better. By treating the same word with different cases as two separate words, we lower the occurrences of both words and reduce the quality of both words' feature vectors and prediction accuracy. Non-English words and website links are usually quite unique and have a relatively low occurrence among the tweets. The Word2Vec does not learn these words well. Considering these links and non-English words are randomly associated with other meaningful words in our Word2Vec mod-

els, we treat these words, stop words, links, and punctuations as noise and we remove all of them from our text corpus.

The last step of our preprocessing work is to find out phrases hidden in our training data. Phrases are valuable resources in our task because a lot of landmark names, celebrity names, restaurant names, and sports team names are phrases and they mean quite differently in separate words. For instance, "Big Apple City" is a nickname for New York city while apple means a kind of fruit, and "Statue of Liberty" is a landmark in New York while statue and liberty are just general nouns. Many phrases give us more geo-information, and in contrast, words tend to provide us less. In our dataset, all phrases, bigrams and trigrams, are formed with Phrases module in Gensim. The preprocessing process for San Francisco is visualized in Figure 3.3.

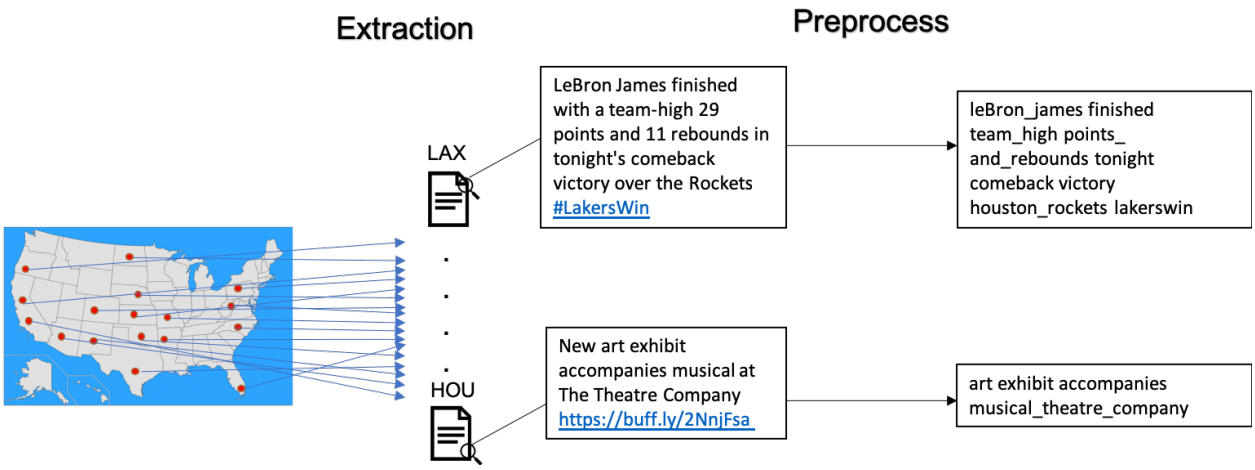

Figure 3.3: Tweet preprocessing process

4. IDENTIFYING GEO-SENSITIVE WORDS

In this chapter, we aim to find geo-sensitive words lying in three big cities and enable comparison for words in all 27 cities. By doing this, we want to understand how geo-sensitive words are related to each other, and how we can find these words among tens of thousands of terms. Our approach begins with training separate Word2Vec models for 27 cities' text corpora and a global Word2Vec model which contains tweets in all text corpora. Based on these models, we align cities' models with the global Word2Vec model by using Orthogonal Procrustes. We validate our aligned Word2Vec models by conducting a common word discovery task. Finally, we find a few geo-sensitive words as seed and look at their nearest neighbors to find more in 4 big cities with adequate tweets and transform them in a geo-sensitive word dataset. The dataset serves our evaluation experiment in Chapter 5.

4.1 Training Geo-Specific Embeddings

Table 4.1: Key hyperparameters in Word2Vec Training

Parameter	Value
Dimensionality of the feature vectors	100
Size of training window	5
Threshold of sampling	1e-3
Minimum word frequency	5
The initial learning rate	0.025
The minimum learning rate	0.0001

To produce Word2Vec models for 27 cities, we train an independent model for each city's corpus with Radim Rehurek Python implementation of Word2Vec in gensim [23]. In our experiment,

we choose to employee Continuous Bag of Words method as our training algorithm and negative sampling as noise contrastive estimation method. The hyperparameters in the training model is listed in Table 4.1. For each city, before the training starts, we tokenize each tweet in the current city's text corpus into a list of tokens and serve a list of lists that contains all tokenized tweets as the input of Word2Vec. The output of Word2Vec is a model that contains a vocabulary constructed based on the input text corpus and learned high dimensional vector representations of words according to the hyperparameters specified. For the global model, we fetch text copora from 27 cities and train it with the same set of hyperparameters employed by city models.

Houston	
Term	cosine similarity
galveston_island	0.847
galveston	0.799
crystal	0.788
banana_bend	0.747
jamaica	0.732
surfside_beach_gulf_coast	0.718

Los Angeles	
Term	cosine similarity
beach_pier	0.744
beach_grand_prix	0.719
huntington_beach	0.672
longbeach	0.661
beach_malibu	0.633
beach_antique_market	0.629

Table 4.2: Geo-Sensitive words in most similar words by feature vector for word "beach" in Houston and Los Angeles

After the training process, we compute the most similar words by feature vector for a few words. For example, as shown in Table 4.2, the most similar words for the word "beach" in Houston and Los Angeles models are mostly local beach names or region names near beaches in the city. This shows that Word2Vec model for a single city is able to capture the geo-information in city text corpus by learning the relationships of between geo-sensitive words in location-specific text corpus. In another example for the word "nba", we find some positive results but also some

problems. The result is shown in Figure 4.3. In Houston, there are a few words that occur to be not related to Houston Rockets, local basketball team, including 'espn', 'lakers', and 'sportscenter'. As we can see, these words still make sense for a high similarity with the word 'nba'. Lakers is famous nationwide team while 'sportscenter' is a sports TV show in 'espn'. While in Los Angeles model, 'Spurs' occurs due to the NBA playoff. 'Knicks' occurs since it's also a popular team nationwide. Lastly, 'lebron_james' is an influential basketball player in the league. These words are too general in both text corpora and need to be treated as noises.

Houston	
Term	cosine similarity
espn	0.847
harden	0.799
james_harden	0.788
dwight	0.747
lakers	0.732
sportscenter	0.718

Los Angeles	
Term	cosine similarity
knicks	0.744
spurs	0.719
cp3	0.672
lebron_james	0.661
kobebryant	0.633
lakersnation	0.629

Table 4.3: Geo-sensitive words in most similar words by feature vector for word "nba" in Houston and Los Angeles

4.2 Aligning Word Embedding

For our application, we want to predict which city a geo-sensitive word is from based on derived word embedding. In order to do this, we have to ensure all word feature vectors from 27 Word2Vec word embeddings are in the same vector space for direct comparison. In the beginning, we test if words' feature vectors can compare directly in different city models. We perform some cosine similarity computations on two feature vectors from two cities for the same word. We compute the cosine similarities for all words that exist in both New York and Chicago. The histogram in

Figure 4.1 of words' cosine similarities shows that word embedding in these two cities are not in the same vector space since most words' meanings (cosine similarities) are not close. Most words' similarities under under 0.4.

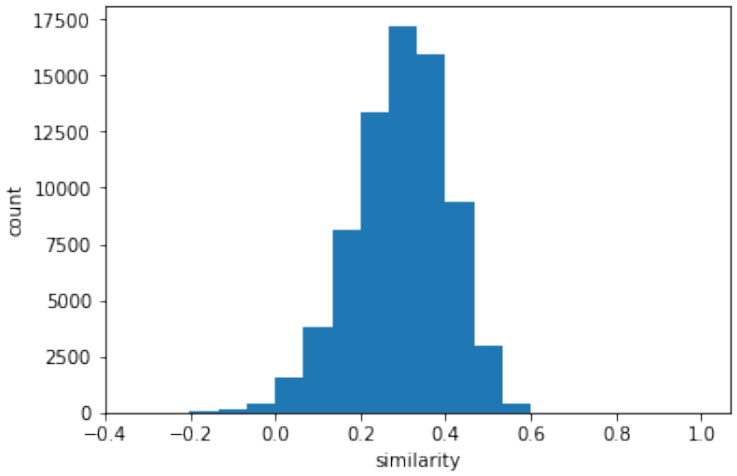

Figure 4.1: The histogram for cosine similarities between words in common between New York and Chicago

By investigating the Word2Vec training algorithm, we find the difference between two independent Word2Vec training processes is a different random seed. This means that all the neural network weights are initialized with random weight. All weights most likely converge to different values in two standalone training processes. The similarity of words in different models stays the same while the absolute position of them changes. This is because, in a Word2Vec model, the similarity of two words is not determined by the absolute positions of these two words, instead, the only thing matters is the cosine similarity between them. Therefore, we need to discover a method to align all word embeddings from different cities into one vector space to let words in two Word2Vec models can perform cosine similarity computation directly. Previous work in detecting statistically significant linguistic shifts [24] overcame this problem when it was trying to align word embedding trained for different time snapshot. The solution is Orthogonal Procrustes [25]. In our work, we adopt the similar approach.

First of all, we use all tweet text corpora in all cities and train them into one global word embedding with Word2vec. This global word embedding has its own vector space and we align our word feature vectors in all cities into this vector space. There are two assumptions for our approach. The first assumption is that we assume that most words in different cities have similar meanings, thus the local relationship between word is preserved. The second one is that we can always find a linear transformation that maps a word from one city word embedding to the global word embedding.

In the next step, we align word embeddings from all cities with the global word embedding with orthogonal Procrustes. Defining $W^{(c)} \in R^{d \times |V|}$ as the matrix of word embeddings learned at city c, we align the current word embedding with the global word embedding while preserving cosine similarities by optimizing:

$$\Re^c = arg \min_{Q_c^t Q_c = I} ||Q_c W^c - W^g||_F$$

where g represents global.

To justify our alignment on word embeddings from every city, we compute the cosine similarity between a feature vector in every city's aligned word embedding and feature vector in global word embedding for the same word. The similarity for each word i is defined as:

$$similarity(w_i^g, w_i^c) = cos - sim(w_i^g, w_i^c) \tag{4.1}$$

The histograms for three cities that contain most tweets are in Figure 4.2. These cities' geo-specific embeddings also give us the best alignment result. From the histograms, we observe that the similarities between these three cities' aligned word embedding and the global word embedding for same words are reasonable. Most words have a cosine similarity between 0.7 to 1.0. Since the assumption is that most words' in all cities will have similar meanings, the alignment result for these three cities is valid. We re-compute the cosine similarities for all words that exist in both New York and Chicago with aligned embeddings in Figure 4.3. We find a high cosine similarities

for all words in these two aligned models with most of them between 0.4 to 0.8.

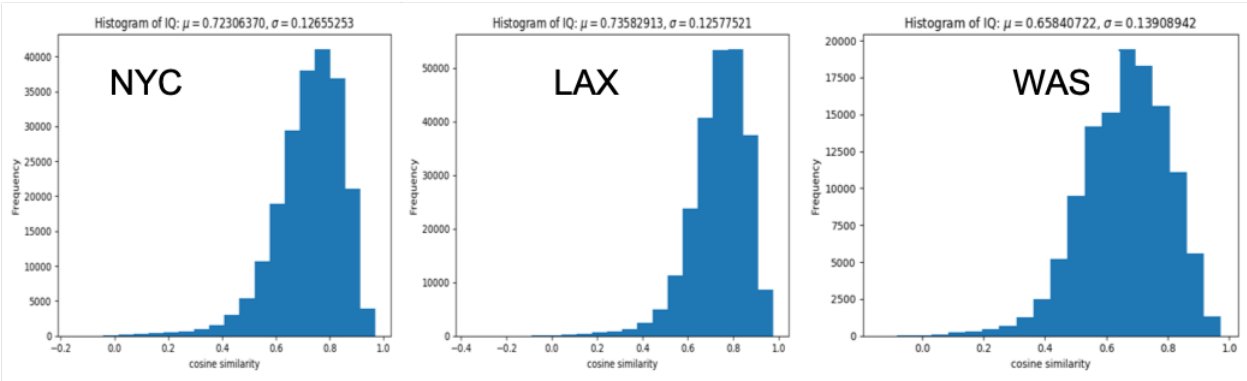

Figure 4.2: The histograms for cosine similarity between a feature vector in New York, Los Angeles, Washington, D.C. city's aligned word embedding and feature vector

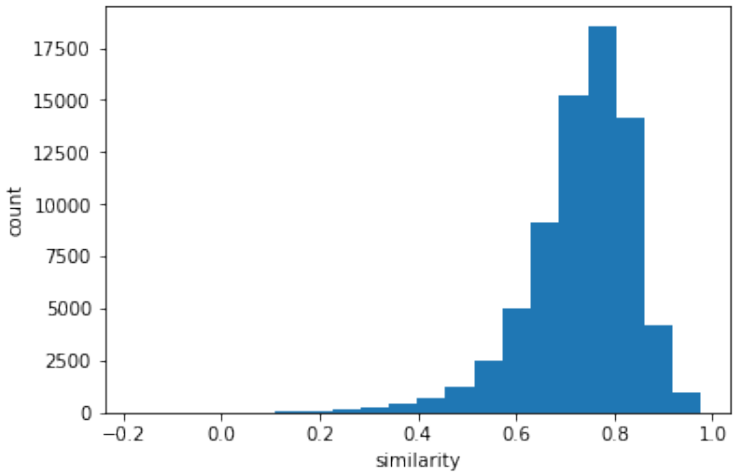

Figure 4.3: The histograms for the cosine similarities for all words that exist in both New York and Chicago with aligned embedding

For the rest of cities, the similarities for words are a little bit lower. We plot the relationship between the number of tweets for each city and their average cosine similarities in Figure 4.4. In this figure, we can clearly see that the similarities are increasing exponentially with the number of

tweets increases. There can be two reasons for this phenomenon. The first reason is that Word2Vec needs a lot of raw text data to build an accurate model. Because of this, cities with less raw text data will have a less precise model which has a negative effect on the alignment process later. The second reason is that cities with more raw text data will bias our global model's word feature vector more than cities with fewer data. Therefore, the cosine similarities for cities with less raw text data have a lower cosine similarity.

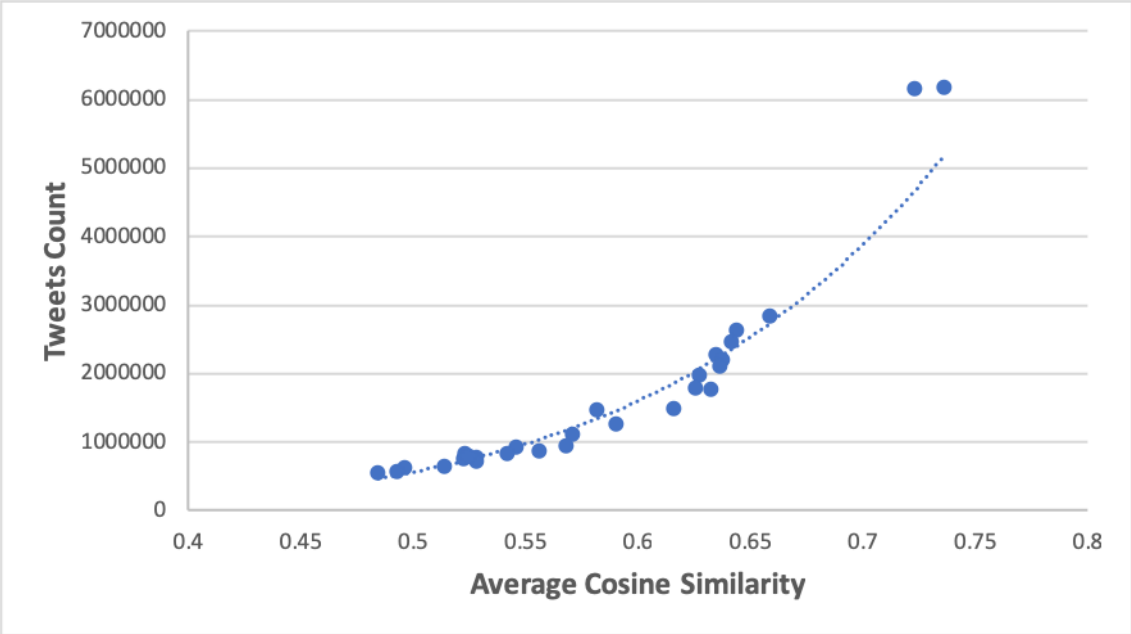

Figure 4.4: Average cosine similarity between feature vectors in every city's aligned word embedding and feature vector in global word embedding for the same word

4.3 Common Word Discovery

To further validate our aligned word embeddings and find out more useful information in it, we decide to find some words that share the same meanings in all cities. For example, since we are using tweets, "tweeted" should clearly be a common word in tweets which means the action that someone published a tweet. "Starbucks" should be the popular coffee shop in the United States. We refer these words as common words. We define common words to be words we use every day

and words used a lot in tweets with no confusion in meaning. In our aligned word embedding, a common word is a word that occurs most frequently in all cities, and its feature vector's cosine similarity in different cities against its global feature vector is high. To accomplish this, we first find top 10000 words which have the highest word frequency in each city. Then we intersect word sets across all cities. Lastly, we compute the cosine similarity using equation (4.1) for each word in the intersection we found earlier. We claim all words that have a cosine similarity higher than 0.5 will be a common word. The result of our discovery is shown in Figure 4.5.

Figure 4.5: Common Word Discovery

We filter out 1636 words. Since the full list of the word is long, we randomly pick up a few words and display them in a word cloud. We read through all these words, and most of them make sense.

City	New York	Los Angeles	Washington, D.C.
# of tweets	6170437	6160402	2850201
# of words and phrases	239798	212884	123528

Table 4.4: Tweets and phrases counts in New York, Los Angeles, and Washington, D.C.

4.4 Geo-Sensitive Word Dataset

In the previous section 4.2, we discover that aligned word embeddings from New York, Los Angeles, and Washington, D.C. have a higher average cosine similarity score compared to other cities, and they are indeed cities with most tweets. The statistics for tweet number and phrase number are in Table 4.4. Therefore, we decide to use geo-sensitive words from these three cities to conduct our classification experiment. Besides this, we also include a set of common words with their feature vectors as our test data. By adding common words to our dataset, we can prove that our classifiers in evaluation section are able to distinguish common words from geo-sensitive words.

To find geo-sensitive words in our word embeddings, we first manually find 90 words in each city that are geo-sensitive words. We call them seed words. Then we look at the top 50 nearest neighbors for all these words. By looking these words in three cities, we find 902 geo-sensitive words in Los Angeles, 1085 words in New York, and 567 words in Washington, D.C. In the last step, we save common word's feature vector in the global Word2Vec model, geo-sensitive words in three cities' Word2Vec models into separate files. In accompany with these feature vectors, we create two types of label for each feature vectors: a numeric label from 0 to 3 , and a four dimension one-hot vector to indicate which city the word is from. We treat common words as a city. $0, 1, 2, 3$ and $[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]$ stand for common words, Los Angeles, New York City and Washington D.C. respectively.

5. EVALUATION

In this chapter, we build a simple location prediction application for geo-sensitve words based on classification to evaluate our geo-sensitive word dataset and our Word2Vec models. We treat common words in the dataset as a city. Given the input of our classifier, a geo-sensitive word feature vector from all four cities, we want to predict which city the geo-sensitive word belongs to. To accomplish this, we adopt four different classifiers. Using 10-fold cross-validation, we measure the performance of each classifier with the metrics described in Section 5.1. By building and evaluating this application, we hope the classifiers to identify the geo-information encoded in geo-sensitive words' feature vector and categorize these words into the correct cities. In addition to this, we discuss the impact of learning models, feature vector dimension, and geo-specific embedding on classifiers' performance.

5.1 Metrics

Table 5.1: Notation Table For Metrics

T	the set of all terms
C	the set of all cities
t	a single term in T
T_c	the set of terms in city c
L_t	the true label for term t
$\hat{L}_t$	the predicted for term t

To evaluate the quality of our classifiers, we compare the predicted city for each word versus the actual city this word belongs to. We report four metrics. We define the notation for our metrics in Table 5.1. The first metric we use is **Accuracy**, which considers the percentage of the number

of terms that are classified to the correct city to all terms. This metric helps us to have a general idea on the overall performance of our classification task. It is defined as:

$$Accuracy(T) = \frac{|\{t|t \in T \wedge L_t = \hat{L}_t\}|}{|T|}$$

To have more insights into our classifiers' performance, we further measure the precision and recall of our results. We define the **Precision For Each City** and **Recall For Each City**. **Precision For Each City** is the ratio of terms are correctly predicted to be in city c to total terms are predicted to be in city c:

$$Precision(T_c) = \frac{|\{t|t \in T_c \wedge L_t = \hat{L}_t\}|}{|\{t|\hat{L}_t = c\}|}$$

Recall For Each City is the ratio of terms that are correctly predicted to be in city c to total terms are actually in city c:

$$Recall(T_c) = \frac{|\{t|t \in T_c \wedge L_t = \hat{L}_t\}|}{|\{t|L_t = c\}|}$$

We then use the unweighted average of **Precision For Each City** and **Recall For Each City** from 4 cities as our overall **Precision** and **Recall**:

$$Precision(T) = \frac{1}{|C|} \sum_{c \in C} Precision(T_c)$$

$$Recall(T) = \frac{1}{|C|} \sum_{c \in C} Recall(T_c)$$

Finally, we define the harmonic mean of **Precision** and **Recall** , F1 score as following:

$$F1(T) = \frac{1}{|C|} \sum_{c \in C} \frac{2Precision(T_c)Recall(T_c)}{Precision(T_c) + Recall(T_c)}$$

In addition to these metrics, we also compute the **Area Under Receiver Operating Characteristic Curve (AUC-ROC)** definition of from prediction scores to calculate the overall performance of the classification model.

5.2 Classifiers

In this section, we explain four different classifiers including a multinomial logistic regression classifier, a support vector machines (SVM) classifier, a random forests classifier, and a multilayer perceptron neural network.

5.2.1 Multinomial Logistic Regression Classifier

Logistic regression is a classic supervised learning model for linear classification. Its simplicity of implementation and usefulness make it the most prevalent and long-lived algorithm for solving industry level problems. Many standard and customized performance metrics can be applied to its probability score and later trimmed for different real world problems. Logistic regression is time and memory efficient which makes it a good choice for training large data online. In addition to this, it is resilient to small noise and multi-collinearity with L2 regularization [26]. In this experiment, we will use a multinomial logistic regression classifier to predict the right location for each geo-sensitive word.

Given the input of our classifier, a geo-sensitive word feature vector $\boldsymbol{x}^i$ from all four cities, we want to predict which city the geo-sensitive word belongs to. For the training data, we use a one-hot vector $\boldsymbol{y}^{(i)}$ to indicate which city the word belongs to.

For each possible class(four cities), our classifier has $\boldsymbol{\omega}^y \in R^n$ where $y \in \{1, ..., n\}$ to decide the Negative Log likelihood of class being y for a input vector using softmax function [27]:

$$-\log p(y|\boldsymbol{x}, \beta) = \log\left(\sum_{y'=1}^{n} e^{\boldsymbol{\omega}^{\boldsymbol{y'}} \cdot \boldsymbol{x}}\right) - \boldsymbol{\omega}^y \cdot \boldsymbol{x} \tag{5.1}$$

$$\text{where } \beta = \{\boldsymbol{\omega}^t\} \text{ for } t \in \{1, ..., n\}$$

We then decide our object of the model. Given of training data $D = \{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}$ for $i \in 1, ..., m$, we want to find $\boldsymbol{\omega}$ that minimizes our negative likelihood of D with cross-entropy loss [27]:

$$minimize \ -\frac{1}{m}\sum_{i=1}^{m} \log p\left(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}, \beta\right) + \frac{1}{2}\boldsymbol{\omega}^{\boldsymbol{T}}\boldsymbol{\omega}$$

where $\frac{1}{2}\boldsymbol{\omega}^{\boldsymbol{T}}\boldsymbol{\omega}$ denotes to the L2 regularization term.

5.2.2 SVM

Support vector machines (SVM) are efficient supervised learning models for classification, regression and outliers detection. Given labeled training data, a SVM model separates two types of categorized data by a hyperplane that maximizes the distance to nearest training data points of any class. A few advantages of SVM include its effectiveness in high dimensional spaces, memory efficiency and high versatility for custom kernels [28]. In this experiment, we will use a multi-class C-Support Vector classifier(C-SVC) [29] with penalty parameter of the error term equals to 1, linear kernel, and "one-against-one" approach [30] for our task.

We will build $4 * (4-1)/2 = 6$ classifiers for each pair of class. Then we will train them to distinguish the word vector from one city to another. Then the final multi-class classification decision is made by maximum voting [30].

Given the input of our classifier, a geo-sensitive word feature vector $\boldsymbol{x}^i$ from all four cities, we want to predict which city the geo-sensitive word belongs to. For the training data, we use numeric label y^i to indicate which city the word is belonging to.

For each distinguished classifier between two cities, given geo-sensitive word feature vectors $\boldsymbol{x_i} \in R^m, i = 1, ..., n$, from two cities, and an indicator vector $\boldsymbol{y} \in R^n$ where $y_i \in 1, -1$, the primal optimization problem is [31]:

$$\min_{\omega,c,\varepsilon} \frac{1}{2}\boldsymbol{\omega}^{\boldsymbol{T}}\boldsymbol{\omega} + \lambda \sum_{i=1}^{n} \varepsilon_i \tag{5.2}$$

$$\text{subject to } y_i(\boldsymbol{\omega}^{\boldsymbol{T}}\phi(x_i) + c) \geq 1 - \varepsilon_i$$

$$\varepsilon_i \geq \text{for all } i$$

where $\phi(\boldsymbol{x_i})$ transforms $\boldsymbol{x_i}$ into a higher-dimensional vector and $\lambda > 0$ is the regularization parameter.

We solve the dual problem to avoid high dimensionality of the vector variable $\boldsymbol{\omega}$ as bellow:

$$\min_{\alpha} \frac{1}{2}\boldsymbol{\beta^T} Q \boldsymbol{\beta} - \boldsymbol{v^T}\boldsymbol{\beta} \tag{5.3}$$

$$\text{subject to } \boldsymbol{y^T}\boldsymbol{\beta} = 0$$

$$0 \leq \beta_i \leq C \text{ for all } i$$

where Q is a n by n matrix with $Q_{ij} \equiv y_i y_j K(x_i, x_j)$ and $\boldsymbol{v}$ is a vector with all ones. $K(\boldsymbol{x_i}, \boldsymbol{x_j}) \equiv \phi(\boldsymbol{x_i})^T \phi(\boldsymbol{x_j})$ is the kernel function. In this experiment, we use radial basis function kernel.

The optimal $\boldsymbol{\omega}$ satisfies

$$\boldsymbol{\omega} = \sum_{i=1}^{n} y_i \beta_i \phi(\boldsymbol{x_i})$$

Finally, the decision function is:

$$sgn\left(\sum_{i=1}^{n} y_i \beta_i K(\boldsymbol{x_i}, \boldsymbol{x}) + b\right)$$

.

5.2.3 Random Forest

Random forest is one of the most common but effective supervised learning algorithms in data science. They can be both used to build models for regression and classification related problems. In general, a random forest model begins with its building block, a decision tree. Given an input, a decision tree will move the input down to the leaf nodes of it based on its condition switch. A random forest, in essence, is an ensemble of many decision trees with different decision-making criteria. The standalone decision trees will act as a "weak learner" in the model while the random forest will combine the decisions from different trees and make a final decision as a "strong learner". In terms of our problem, every single decision tree will predict a target class a geo-sensitive word is belong to, and the random forest will make the final prediction based on results gained from trees in its "forest". In contrast to other classification algorithms, the random forest will never run into the trap of overfitting, and it is capable of handling missing values [32].

In our experiment, the input is a set of geo-sensitive word feature vectors from Word2Vec, we want to predict which city the geo-sensitive word belongs to.

There are two phases in random forest classification. In the first stage, we start with random forest creation [32]:

Algorithm 1: Random forest creation algorithm

```
for a certain number of trees do
    for a certain number of nodes do
        Pick k features from all 100 features (dimension of word feature vectors)
        Use the selected features and compute a node based on optimal splitting strategy
        Further divide these nodes following the same splitting strategy
        Create a node
    end
    create a random decision tree
end
```

In the second phase, we will predict the target class. We will use all the decision tree created in the first phase to predict a vote for each input x_i. The highest vote result will be the final prediction [32]. The random forest in this experiment is with the maximum depth of the tree equals to 10 and the number of trees for estimation equals to 300.

5.2.4 Neural Network

An artificial neural network has become extremely popular in recent years with the occurrence of the backpropagation algorithm. Inspired by the human brain, an artificial neural network is composed with a set of interconnected artificial neurons. There are usually three layers for a basic neural network: an input layer, a hidden layer, and an output layer. The input layer is responsible for fetching data and transmitting them to the hidden layer. Then, the hidden layer will perform some computation on the activation function in the neurons and send it to the output layer. Based on this simple model, a lot more complex and advanced neural network models have been created to fit more specific and customized problems in different research fields and business solutions

including convolutional neural networks for image recognition [33], Boltzmann machine networks for collaborative filtering [34], long short-term memory (LSTM) networks for natural language processing (NLP) [35]. For this experiment, we endorse a simple three-layer neural network to this classification task. We will have 100 units for our input layer. We will then have dense layer with 20 rectified linear units collect the essential geo-sensitive information from all inputs as our hidden layer. Finally we will use softmax function on top of a dense layer with 4 units to normalize the output vector to a probability distribution. The 4 units in the output layer represent the possibility of the geo-sensitive word be in four different cities.

Given the input of our classifier, a geo-sensitive word feature vector $\boldsymbol{x}^i$ where $i \in \{1, ..., n\}$ from all four cities, we want to predict which city the geo-sensitive word belongs to. For the training data, we use a one-hot vector $\boldsymbol{y}^{(i)}$ where $\boldsymbol{y}^{(i)} \in \{0, 1, 2, 3\}$ to indicate which city the word is belonging to.

The output probability for each input $\boldsymbol{x}^{(i)}$ for be in 4 class is:

$$p(\boldsymbol{x}^{(i)}) = softmax(\boldsymbol{w_2}\boldsymbol{\delta}(\boldsymbol{w_1}\boldsymbol{x}^{(\boldsymbol{i})} + \boldsymbol{b_1}) + \boldsymbol{b_2})$$

where $\boldsymbol{w_1}, \boldsymbol{w_2}$ are weights, and $\boldsymbol{b_1}, \boldsymbol{b_2}$ is the bias terms. The $\delta(x)$ is the activation function.

The objective function for us to minimize with cross-entropy loss is:

$$-\sum_{0}^{3} \boldsymbol{y}^{(\boldsymbol{i})} \log\left(p(\boldsymbol{x}^{(i)})\right)$$

5.3 Experimental Results

In this section, we detail an experimental study of geo-sensitive word location prediction with a variety of learning algorithms and different feature vector dimensions. We build our models based on the same dataset. The goal of our experiment is to understand: (i) which learning model yields the best performance for words' location prediction; (ii) how does the dimension of feature vectors affects the classification performance; and (iii) are geo-specific embeddings more effective than

global embeddings for location prediction task?

5.3.1 Impact of Learning Algorithm

Classifier	LR	SVM	Random Forest	Neural Network
Accuracy	88.04%	90.14%	93.16%	**94.22%**
Precision	87.75%	89.36%	92.22%	**94.56%**
Recall	87.16%	89.53%	90.03%	**93.80%**
F1	87.18%	89.33%	90.64%	**94.14%**
AUC-ROC	91.63%	93.16%	93.75%	**96.13%**

Table 5.2: Performance metrics for learning algorithms

We begin by investigating the impact of learning models. We use the feature vector with 100-dimension to conduct our experiment. We exam four different classifiers including a multinomial logistic regression classifier, a support vector machines (SVM) classifier, a random forests classifier, and a multilayer perceptron neural network. All the hyperparameters for each classifier and experiment setup are described in Section 5.2. The experiment result is shown in Table 5.2. In this experiment, all four learning algorithms reach a high accuracy above 85%. Based on the fact that logistic regression classifier is a linear model and SVM classifier used a linear kernel function, we confirm that most geo-sensitive words' feature vectors in our dataset are linearly separable. The AUC-ROC scores indicates that all classifiers are capable of distinguishing between cities.

The recall and precision for all models are relatively close to their F1 score except the random forest model. To investigate this, we compute recall and precision of each city. The recall for each city is shown in Figure 5.1. From this figure, we observe that random forest's recall for Washington is much lower than other three cities. In contrast to the recall for each city, the precision for each city in Figure 5.2 exhibits quite different values. The precision for Washington is extremely

high. This indicates that even Random Forest algorithm can not find all geo-sensitive words for Washington, but it is capable to make all its retrieved instance correct. Since the precision and recall for common words are all high, the value difference between recall and precision for random forest clearly of three remaining cities proves that some words in Washington are characterized as geo-sensitive words from Los Angeles or New York.The recalls of Los Angeles and New York are high since they include most words belongs to them plus the extra words from Washington. However, when it comes to precision, since these cities contain extra words from Washington, all their precisions become lower. The possible explanation to this phenomenon is that we have unequal amount of words for these three cities. The classifiers are biased to cities contain more words. We can also spot similar behavior for other three learning models where this phenomenon is clearer in Random Forest and logistic regression classifier. SVM and neural network, on the other side, are more resilient to the inequality of words from different cities.

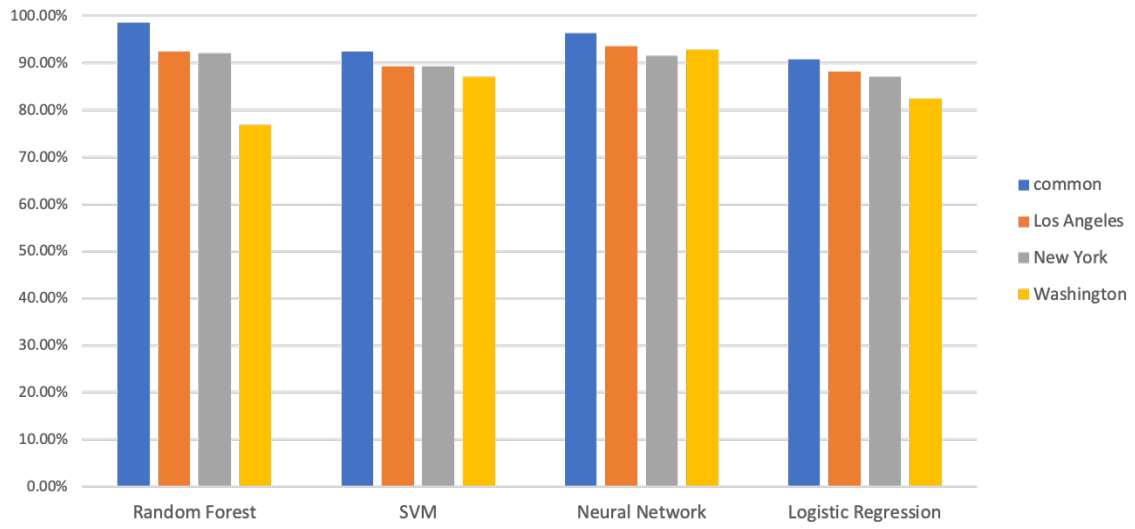

Figure 5.1: Recall for each city

In the end, we conclude that the neural network algorithm provides the best overall performance among four learning algorithms. The neural network algorithm provides the highest accuracy, strongest ability to distinguish between classes and best resilience to inequality of words from

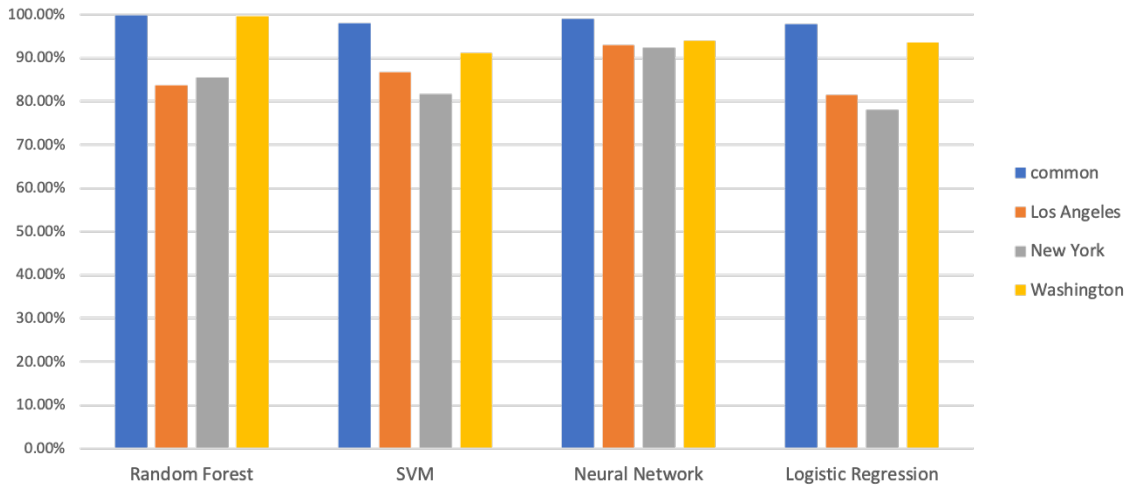

Figure 5.2: Precision for each city

different cities.

5.3.2 Impact of Feature Vector Dimension

Given the good performance of our classifiers, we further study the impact of feature vector dimension. In this experiment, we train 5 sets of word embeddings with different feature vector dimensions. The dimensions we chose for each set includes 10, 50, 100, 200, and 400. Each set contains a global word embedding and 27 geo-specific embeddings for each city with the specified dimension. Five geo-sensitive word datasets are re-constructed with the same set of words but different feature vector dimensions.

To illustrate the impact of an increasing dimension, we plot the accuracy, F1 and AUC-ROC score in Figure 5.3 for 4 classifiers. We observe that the accuracy, F1-score, and AUC-ROC are enhanced while the feature vector dimension is increasing. The largest improvement happens when the dimension changes from 10 to 50. The multinomial logistic regression classifier, the support vector machines (SVM) classifier, and the neural network model has an clear improvement with +8.95% in accuracy, +10.41% in F1 score, and +6.82% in AUC-ROC score. From 100 to 400, all the metrics nearly stop growing except for random forest. The best performance for all classifiers are obtained at dimension 400. With the increasing size of feature vector, more information is

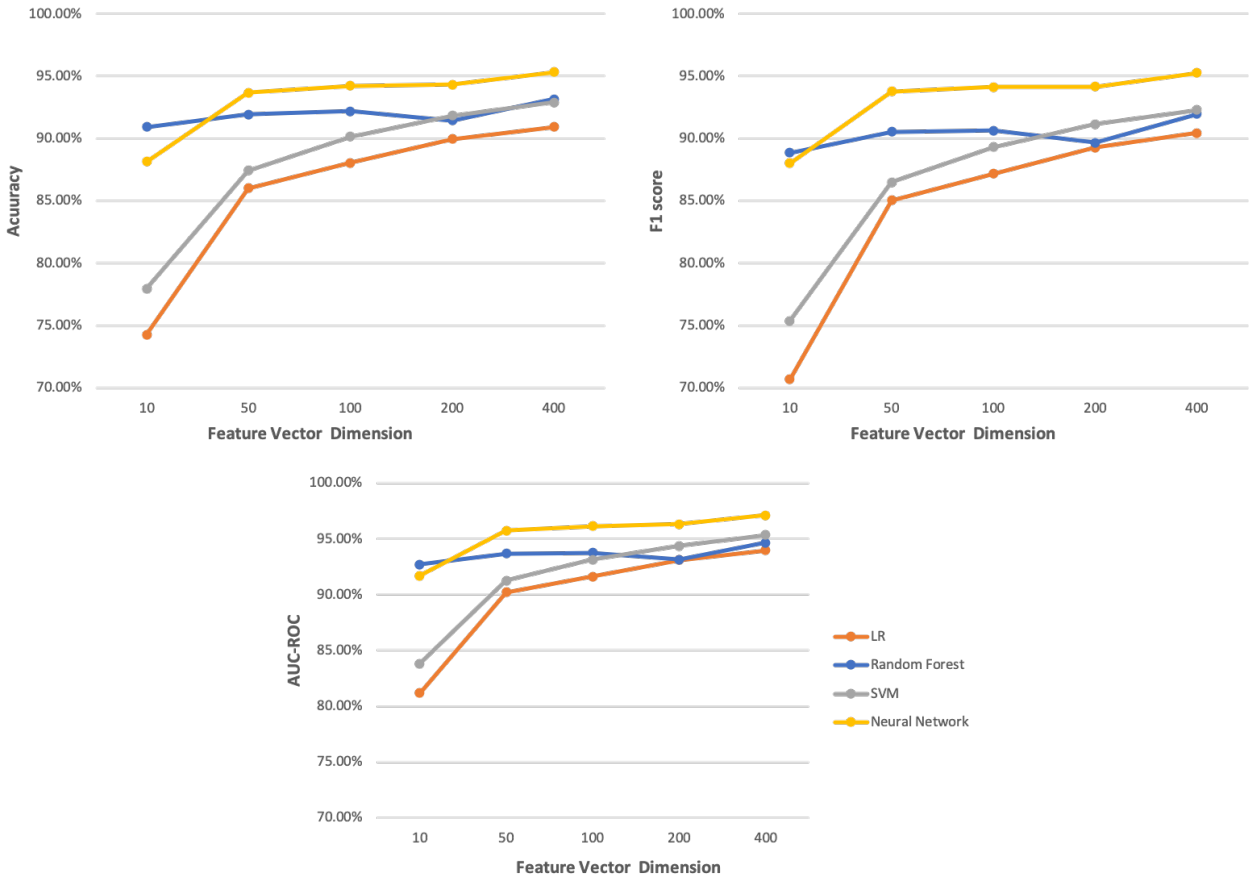

Figure 5.3: Accuracy, F1 score and AUC-ROC score for different feature vector dimension

learned in the Word2Vec model which helps the classifiers to make better decisions. we conclude that providing more information to the classifiers helps them to improve their performance.

We further investigate if feature vector dimension increasement has positive impact on resilience to inequality of words from different cities. As we can see, both precision and recall of all classifiers are increasing with the feature vector dimension grows. However, the unbalance between precision and recall caused by the inequality of words is not improved. The most unbalanced classifier is still random forest. We conclude that the feature vector dimension increasement does not help classifiers to improve inequality of words.

LR	dim-10	dim-50	dim-100	dim-200	dim-400
Precision	71.11%	85.54%	87.75%	89.57%	90.50%
Recall	70.66 %	84.98%	87.16%	89.52%	90.88%
SVM	dim-10	dim-50	dim-100	dim-200	dim-400
Precision	77.70%	86.83%	89.36%	91.11%	91.86%
Recall	74.77%	86.59%	89.53%	91.44%	92.97%
Random Forest	dim-10	dim-50	dim-100	dim-200	dim-400
Precision	90.15%	91.77%	92.22%	91.67%	93.21%
Recall	88.39%	89.98%	90.03%	89.09%	91.55%
Neural Network	dim-10	dim-50	dim-100	dim-200	dim-400
Precision	90.61%	94.69%	94.56%	94.23%	94.97%
Recall	85.89%	92.98%	93.80%	94.24%	95.65%

Table 5.3: Precision and recall for different feature vector dimension

5.3.3 Impact of Geo-Specific Embeddings

An important question remains: have the geo-specific embeddings outperforms the global embeddings? Does it offer a big improvement to our classification task? In previous sections, all experiments are focused on testing the hyperparameters and learning algorithms. If we can use global embeddings to achieve the same performance, it will defeat our purpose of learning the geo-specific embeddings.

To illustrate the impact of geo-specific embeddings, we conduct another set of experiments. For all geo-sensitive words in our dataset, we use feature vectors with 200 and 400 dimension from global word embeddings. We compare the performance of it with our feature vectors from geo-specific embeddings. The result of our experiment is displayed in Figure 5.4, 5.5, 5.4 and 5.5. As we can see, the highest performance gain we have is for random forest with 10.50% improvement on accuracy, 12.78% on F1, and 7.70% on AUC-ROC score with 400 dimension word embedding.

Logistic regression on the other side, has the lowest performance gain with 2.98% improvement on accuracy, 3.61% on F1, and 2.54% on AUC-ROC score with 400 dimension word embedding. The low performance gain on logistic regression symbols a high possibility that the full dataset is not linearly separatable. Even logistic regression can separate most of the words in the dataset, it struggles to separate the remaining portion of words. From these data, we conclude that geo-specific embeddings have a strong positive impact on our classification task. By learning geo-specific embeddings for each city, we are able to capture more geo-sensitive information in word feature vectors.

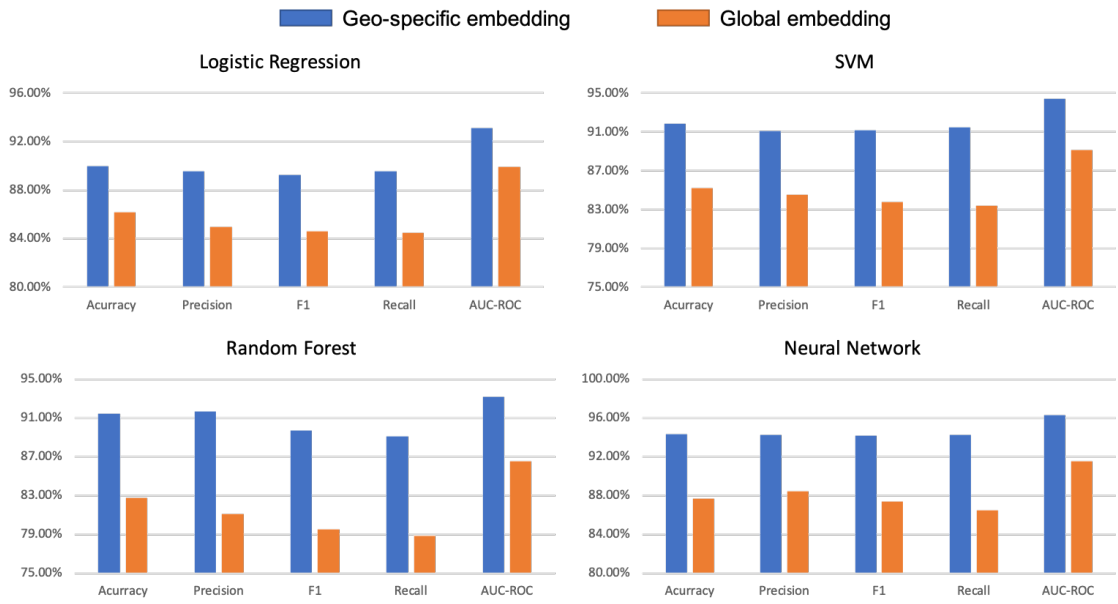

Figure 5.4: Performance comparison between global and geo-specific embeddings with 200 dimension

	LR	SVM	Random Forest	Neural Network
Accuracy	4.41%	7.83%	10.50%	7.60%
Precision	5.46%	7.78%	13.09%	6.57%
F1	5.51%	8.80%	12.78%	7.83%
Recall	5.97%	9.71%	13.00%	8.97%
AUC-ROC	3.57%	5.89%	7.70%	5.21%

Table 5.4: Performance gain for geo-specific embeddings with 200 dimension

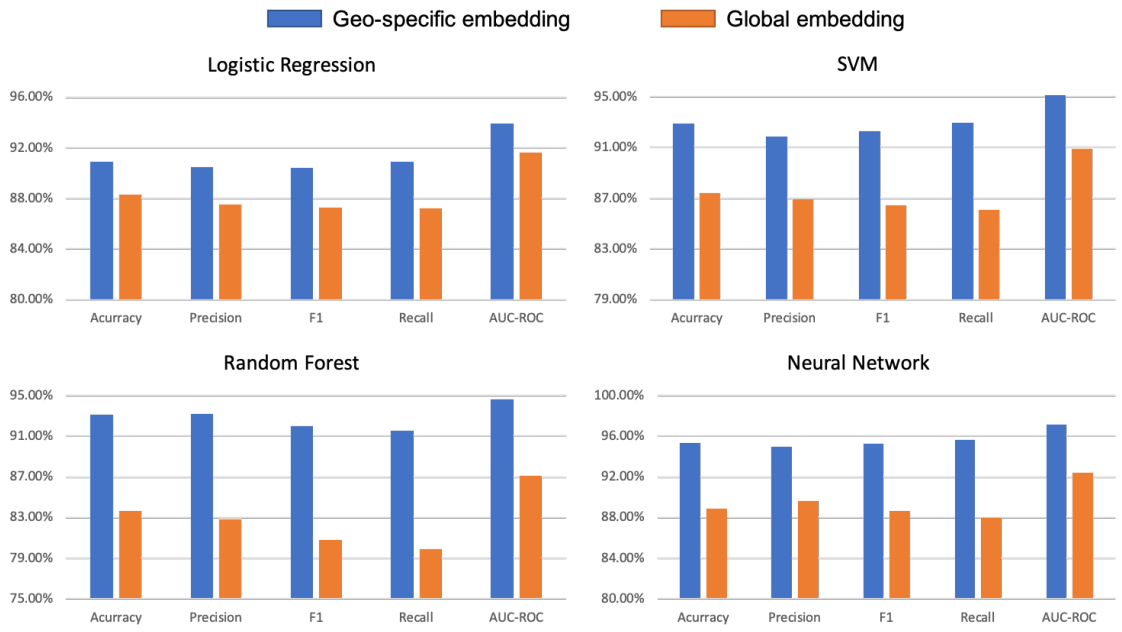

Figure 5.5: Performance comparison between global and geo-specific embeddings with 400 dimension

	LR	SVM	Random Forest	Neural Network
Accuracy	2.98%	5.08%	11.32%	7.22%
Precision	3.39%	4.78%	12.53%	5.95%
F1	3.61%	5.46%	13.85%	7.42%
Recall	4.18%	6.21%	14.58%	8.72%
AUC-ROC	2.54%	3.87%	8.62%	5.07%

Table 5.5: Performance gain for geo-specific embeddings with 400 dimension

6. CONCLUSION AND NEXT STEPS

6.1 Conclusion

The success of leveraging geolocation creates new opportunities for new exciting purpose. As one of the most critical components in geolocation related information, geo-sensitive words is able to deliver multiple benefits for local searches and personalized recommendation systems. To identify these words, our proposed framework overcomes the challenges with the following key contributions: (i) a publicly-available dataset containing geo-tagged English tweets from 27 cities in the United States; (ii) a concrete approach to align separately trained word embeddings with Orthogonal Procrustes; (iii) and a well-rounded evaluation framework for geo-sensitive words. We've aligned all geo-specific embeddings to enable direct comparison between word embeddings. We've discovered over 3000 geo-sensitive words in three large cities in USA. We've showed that our geo-sensitive word classification task with geo-specific embeddings yields an average 7.5% accuracy gain compared to the classification with global word embedding. The classifier with neural network has reached a high accuracy at 95.32%. We have seen that the increasement of feature vector dimension improved the prediction's performance.

6.2 Next Steps

In this study, we have only studied the models for word in different locations. We anticipate continuing work on prediction of user location based on their tweets. We are also interested in fully automating the geo-sensitive word discovery framework by adopting Backstrom's model for analyzing the geographic distribution of terms in search engine query logs [2]. Ultimately, we expect our model can be used to recommend local restaurants, experts, and tourist places based on the users' preference in their living place.

REFERENCES

[1] Z. Cheng, J. Caverlee, and K. Lee, "You are where you tweet: a content-based approach to geo-locating twitter users," in *Proceedings of the 19th ACM international conference on Information and knowledge management*, pp. 759–768, ACM, 2010.

[2] L. Backstrom, J. Kleinberg, R. Kumar, and J. Novak, "Spatial variation in search engine queries," in *Proceedings of the 17th international conference on World Wide Web*, pp. 357–366, ACM, 2008.

[3] B. Han, P. Cook, and T. Baldwin, "Text-based twitter user geolocation prediction," *Journal of Artificial Intelligence Research*, vol. 49, pp. 451–500, 2014.

[4] L. Chi, K. H. Lim, N. Alam, and C. J. Butler, "Geolocation prediction in twitter using location indicative words and textual features," in *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, pp. 227–234, 2016.

[5] W. Niu, Z. Liu, and J. Caverlee, "On local expert discovery via geo-located crowds, queries, and candidates," *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, vol. 2, no. 4, p. 14, 2016.

[6] L. Backstrom, E. Sun, and C. Marlow, "Find me if you can: improving geographical prediction with social and spatial proximity," in *Proceedings of the 19th international conference on World wide web*, pp. 61–70, ACM, 2010.

[7] P. D. Turney and P. Pantel, "From frequency to meaning: Vector space models of semantics," *Journal of artificial intelligence research*, vol. 37, pp. 141–188, 2010.

[8] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[9] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing*

(EMNLP), pp. 1532–1543, 2014.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[11] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[12] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," *arXiv preprint arXiv:1309.4168*, 2013.

[13] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang, *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," *arXiv preprint arXiv:1602.06023*, 2016.

[14] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *arXiv preprint arXiv:1603.01360*, 2016.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," pp. 3111–3119, 2013.

[16] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau, "Sentiment analysis of twitter data," in *Proceedings of the workshop on languages in social media*, pp. 30–38, Association for Computational Linguistics, 2011.

[17] Z. Akata, S. Reed, D. Walter, H. Lee, and B. Schiele, "Evaluation of output embeddings for fine-grained image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2927–2936, 2015.

[18] F. Diaz, B. Mitra, and N. Craswell, "Query expansion with locally-trained word embeddings," *arXiv preprint arXiv:1605.07891*, 2016.

[19] E. Amitay, N. Har'El, R. Sivan, and A. Soffer, "Web-a-where: geotagging web content," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 273–280, ACM, 2004.

[20] S. Chandra, L. Khan, and F. B. Muhaya, "Estimating twitter user location using social interactions–a content based approach," in *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pp. 838–843, IEEE, 2011.

[21] S. Kinsella, V. Murdock, and N. O'Hare, "I'm eating a sandwich in glasgow: modeling locations with tweets," in *Proceedings of the 3rd international workshop on Search and mining user-generated contents*, pp. 61–68, ACM, 2011.

[22] Z. Cheng, J. Caverlee, H. Barthwal, and V. Bachani, "Who is the barbecue king of texas?: a geo-spatial approach to finding local experts on twitter," in *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pp. 335–344, ACM, 2014.

[23] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, (Valletta, Malta), pp. 45–50, ELRA, May 2010. `http://is.muni.cz/publication/884893/en`.

[24] V. Kulkarni, R. Al-Rfou, B. Perozzi, and S. Skiena, "Statistically significant detection of linguistic change," in *Proceedings of the 24th International Conference on World Wide Web*, pp. 625–635, International World Wide Web Conferences Steering Committee, 2015.

[25] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[26] D. G. Kleinbaum and M. Klein, *Logistic regression: a self-learning text*. Springer Science & Business Media, 2010.

[27] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[28] N. Cristianini, J. Shawe-Taylor, *et al.*, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[29] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, ACM, 1992.

[30] S. Knerr, L. Personnaz, and G. Dreyfus, "Single-layer learning revisited: a stepwise procedure for building and training a neural network," in *Neurocomputing*, pp. 41–50, Springer, 1990.

[31] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[32] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[34] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*, pp. 791–798, ACM, 2007.

[35] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.